**International Journal of Advanced Education and Research**
**ISSN: 2455-5746, Impact Factor: RJIF 5.34**
**www.newresearchjournal.com/education**
**Volume 1; Issue 3; March 2016; Page No. 12-17**

# FFT implementation with fused floating-point operations

**[1] Mr. A. Ramakrishna, [2] B. Kavya, [2] K. Ganesh, [2] N. Gayathri, [2] P. Ramya Sudha, [2] D. Satish**

[1] Associate professor, ECE department, VITS College of engineering, Andhra Pradesh, India

[2] Student, ECE department, VITS College of engineering, Andhra Pradesh, India

**Abstract**
This paper describes two fused floating-point operations and applies them to the implementation of fast Fourier transform (FFT) processors. The fused operations are a two-term dot product and an add-subtract unit. The FFT processors use "butterfly" operations that consist of multiplications, additions, and subtractions of complex valued data. Both radix-2 and radix-4 butterflies are implemented efficiently with the two fused floating-point operations. When placed and routed using a high performance standard cell technology, the fused FFT butterflies are about 15 percent faster and 30 percent smaller than a conventional implementation. Also the numerical results of the fused implementations are slightly more accurate, since they use fewer rounding operations. Index Terms—Floating-point arithmetic, fused floating-point operations, fast Fourier transform, Radix-2 FFT butterfly, Radix-4 FFT butterfly.

**Keywords:** FFT, fused floating-point

## Introduction

FLOATING-POINT arithmetic provides a wide dynamic range, freeing special purpose processor designers from the scaling and overflow/underflow concerns that arise with fixed-point arithmetic.

Use of the IEEE-754 standard 32-bit floating-point format [1] also facilitates using the fast Fourier transform (FFT) processors as coprocessors in collaboration with general purpose processors. This paper is concerned with the efficient floating-point implementation of the butterfly units, that in FFT processors. Since many signal processing applications need high throughput more than low latency, the primary focus of the fused elements is to reduce the circuit area with secondary attention to reducing the delay. Two fused floating-point primitive operations have been developed recently to reduce the delay and area of FFT computation units. The first of the fused operations is a fused floating- point two-term dot product unit (Fused DP). The Fused DP is an extension of the fused multiply-add (FMA) operation that was developed initially for the IBM RS/6000 processor [2], [3] and recently added to IEEE Std-754 [1]. The floating-point FMA has several advantages over discrete floating-point adders and multipliers in a general purpose processor. The FMA reduces the latency of a multiplication followed by an addition. Also, a single FMA may be used to replace the floating-point adder and the floating-point multiplier in a system. Some DSP algorithms have been rewritten to take advantage of the presence of FMA units. For example, a radix-16 FFT algorithm that speeds up FFTs in systems with FMA units is described in [4]. The second of the fused operations is a fused add-subtract unit (Fused AS). In FFT algorithms, both the sum and the difference of a pair of operands are needed frequently. In a fused implementation, the sum and the difference operations can share a substantial amount of operand alignment logic with the benefit of reducing the circuit area.

In traditional implementations with discrete floating-point adders and multipliers, the dot product and add-subtract operations may be performed serially, which is relatively slow. Generally, they are performed in parallel, which is expensive both in silicon area and in power consumption, but which provides the best throughput. In this paper, parallel versions implemented with discrete floating point adders and multipliers are used as the baselines. In view of the limited precision required for most DSP applications, all implementations in this paper (discrete as well as fused) support only the 32-bit IEEE-754 standard format [1]. Although a hardware implementation of sub normal is not provided, all four IEEE rounding modes are supported. This facilitates interfacing to other system elements that support the IEEE floating- point standard format.

Section 2 describes a radix-r FFT processor to provide a context for the FFT butterfly units. The next two sections describe the Fused DP and Fused AS units. Subsequent sections describe the use of these two operations to implement FFT butterfly units.

## 2 Fast Fourier Transform Processor

The basic architecture of a radix-r pipeline FFT processor is well known [5]. The basic structure is shown on Fig. 1. It consists of an Inter level cascade of two types of elements. These are radix-r computational elements (identified as CE on the figure) and data reordering elements [6] (identified as RE on the figure). The data flow is in one direction along the heavy lines, which convey r complex data. The light lines carry r _ 1 complex twiddle factors from the TF units (identified as TF on the figure) to the computational elements. The twiddle factors may be computed or may be obtained from a memory. In the data flow, the computational element is used first, then the two types of elements (RE and CE) alternate ending with a computational element. There are two types of computational elements (also called butterfly units). Decimation In Time (DIT) computational elements consist of complex multiplication(s) followed by a sum and difference network. Decimation In Frequency (DIF) computational elements consist of a sum and difference network followed by complex

multiplication(s). As explained in [5], the radix-r computational element computes a Fourier transform of the r-complex inputs that it receives. Use of N radix-r computational elements and N _ 1 reordering elements produces a r N-point FFT or inverse FFT. The data rate is r times the clock rate since on each clock cycle r data enter and r data exit from each element.

## 3 Fused Floating-Point Two- Term Dot Product Unit

The floating-point two-term fused dot product (Fused DP) unit computes a two- term dot product

$$X = AB \pm CD.$$

Although a conventional dot product adds the two products as Shown in (1), the Fused DP unit also allows forming the difference of the two products, which is useful in implementing complex multiplication. The Fused DP unit is based on the fused multiply-add unit shown in the above fig. The Fused dot product unit is shown in Fig. It adds a second multiplier tree, and a revised exponent comparison circuit (shown in Fig. 4) to the FMA. From the carry save adder onward the FDP and FMA are identical. There is a significant area reduction compared to a conventional parallel discrete implementation of two multipliers and an adder, since the rounding and normalization logic of both of the multipliers are eliminated. Designs of a discrete two-term dot product unit constructed

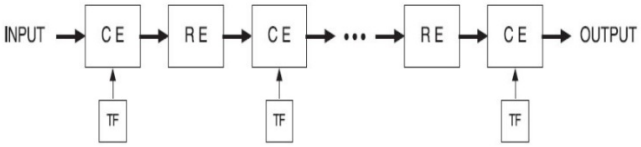
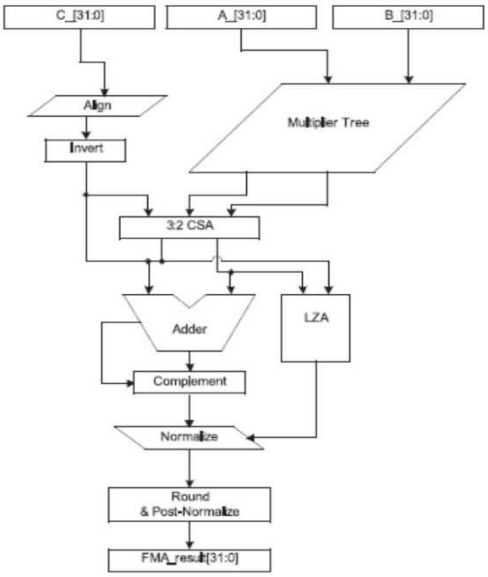
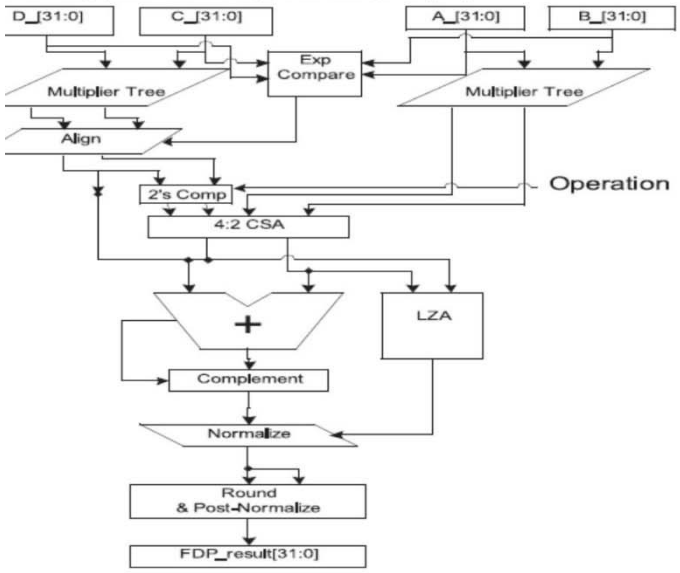



**Fig 1:** Pipeline fast fourier transform processor

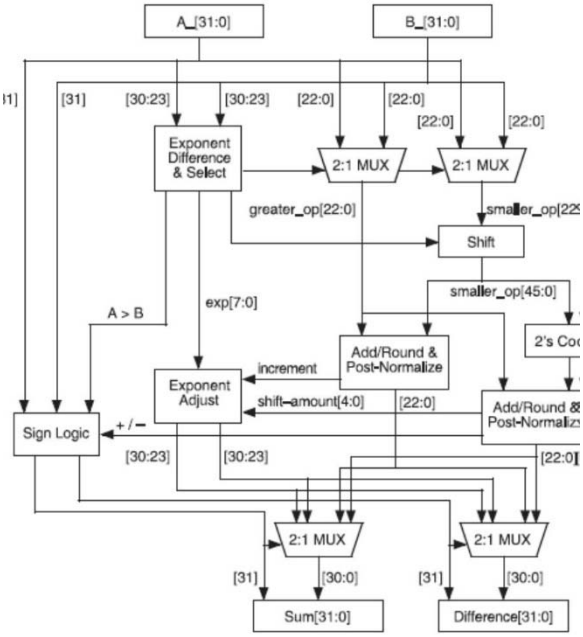



2. Floating-point fused multiply-add unit (after [2-3]).




3. Floating-point fused dot-product unit [7].

with two floating-point multipliers and a floating-point adder and an extracted and back-annotated net list. The dot product implementation results are shown in Table 1.Since the rounding, normalization, and output circuits of the two multipliers are eliminated in the fused design, it is about 33 percent smaller and the delay is about 16 percent less than the discrete




5. Floating-point fused add-subtract unit (after [10]).

parallel implementation. In addition, the Fused DP unit provides slightly more accurate results because only one rounding operation is performed compared to the three rounding operations (one at the output of each multiplier and the other at the output of the adder) of the discrete implementation.

**4 Fused Floating-Point Add- Subtract Unit**

The floating-point fused add-subtract unit (Fused AS) performs an addition and a subtraction in parallel on the same pair of data

$$X = A + B \text{ and}$$
$$Y = A - B.$$

The fused add-subtract unit is based on a conventional floating point adder [8]. Although higher speed adder designs are available (see [9] for example), the basic design shown here serves to demonstrate the concept. A block diagram of the fused add subtract unit is shown in Fig. 5 (after the initial design from [10]). Some details, such as the LZA and normalization logic are omitted here to simplify the figure. The exponent difference calculation, significand swapping, and the significand shifting for both the add and the subtract operations are performed with a single set of hardware and the results are shared by both the operations. This significantly reduces the required circuit area. The significand swapping and shifting is done based solely on the values of the exponents (i.e., without comparing the significands). As a result, if

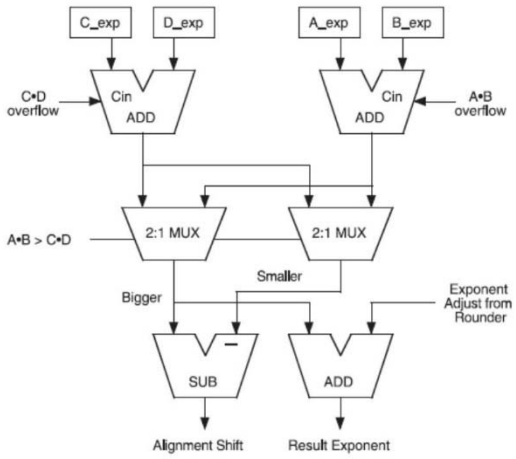



Fig. 1. Pipeline fast Fourier transform processor.
Fig. 2. Floating-point fused multiply-add unit (after [2-3]).
Fig. 3. Floating-point fused dot-product unit [7].
Fig. 4. Exponent comparison circuit. TABLE 1

Fig. 4. Exponent comparison circuit.

TABLE 1
Discrete (Parallel) and Fused Two-Term Dot Product Unit Comparison

| 2-Term Dot Product | Discrete | Fused |
|---|---|---|
| Standard Cell Area | 24,043 $\mu m^2$ | 16,104 $\mu m^2$ (67%) |
| Worst-Case Delay | 3.23 ns | 2.72 ns (84%) |

TABLE 3
Discrete Parallel and Fused Add-Subtract Comparison

| Add-Subtract Unit | Discrete | Fused |
|---|---|---|
| Standard Cell Area | 7,622 $\mu m^2$ | 5,947 $\mu m^2$ (78%) |
| Worst-Case Delay | 1.64 ns | 1.72 ns (105%) |

Discrete (Parallel) and Fused Two-Term Dot Product Unit Comparison the exponents are equal, the smaller significand may be misidentified as the larger operand.

Within the unit, it is advantageous to treat both operands as positive, as shown in Fig. 5, in order to simplify the addition and the subtraction operations. For the sum, the result may overflow by 1-bit; in this case, the Round and Normalize unit downshifts the significand and sends an increment flag to the exponent adjust unit. For the difference, catastrophic cancellation is possible, so 5 bits are passed to the exponent adjust unit. Since the difference may be either positive or negative (depending on which significand is larger), its sign is also passed to the sign logic to create the correct final sign of the difference. The sign logic also controls the multiplexers that select the correct exponents and significands for the sum and difference outputs. To evaluate the benefits of the fused add-subtract unit, a pair of conventional floating-point adders and a fused floating-point add-subtract unit were characterized, using the same process and methodology that was used for the Fused DP. Although a preliminary design based on a different (relatively low speed) 45 nm standard cell library was presented in [10], the design presented here was done with the same cell library that was used for all of the designs in this paper. The area of the main functional elements of a single conventional floating-point adder and the fused add-subtract unit are shown in Table 2. The unpack, exponent difference,

significand swapping and alignment circuits are about the same size for a single adder or for the fused AS unit. The add and subtract circuits are about 50 percent larger for the fused add-subtract unit. The remaining circuits for normalizing, rounding, post normalizing, and packing the results are about twice as big for the fused unit as those of a single floating point adder. In the implementation characterized on Table 2, the post normalization allows either result to have catastrophic cancellation. As indicated above, an optimized design would treat both operands as positive simplifying the normalization logic for the two adders at the minor expense of some sign logic and the requirement for multiplexers to drive the output registers. Another change that should be made in optimizing the design is to compute the round and sticky bits while performing the shift of the smaller significand, so that only 26 bits are used to represent it. This will allow smaller adders to be used, which should reduce the delay. The characteristics of the conventional parallel discrete add subtract unit and the fused unit (Fused AS) are shown on Table 3. The Fused AS unit is approximately 5 percent slower (due to the two's complement operation for the subtraction, heavier loading, and longer lines), but provides about a 22 percent savings in cell area compared to the parallel discrete add and subtract unit.

TABLE 2
Discrete (Parallel) and Fused Add-Subtract Area Comparison

| Function | One Adder | Fused Add-Subtract |
|---|---|---|
| Unpack & Align | 1,404 $\mu m^2$ | 1,482 $\mu m^2$ (106%) |
| Significand Adder | 738 $\mu m^2$ | 1,138 $\mu m^2$ (154%) |
| Normalizer | 989 $\mu m^2$ | 1,924 $\mu m^2$ (195%) |
| Round & Output | 680 $\mu m^2$ | 1,403 $\mu m^2$ (206%) |
| TOTAL AREA | 3,811 $\mu m^2$ | 5,947 $\mu m^2$ (156%) |

## 5 Radix-2 FFT butterfly

To demonstrate the utility of the Fused DP and Fused AS units for FFT implementation, FFT butterfly unit designs using both the discrete and the fused units have been made. First, a radix-2 decimation in frequency FFT butterfly was designed. It is shown in

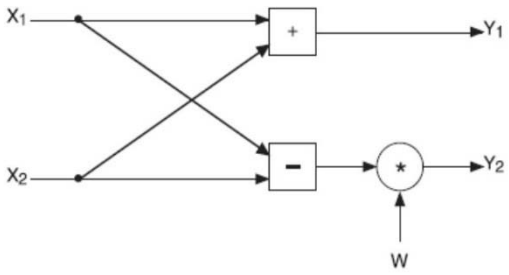
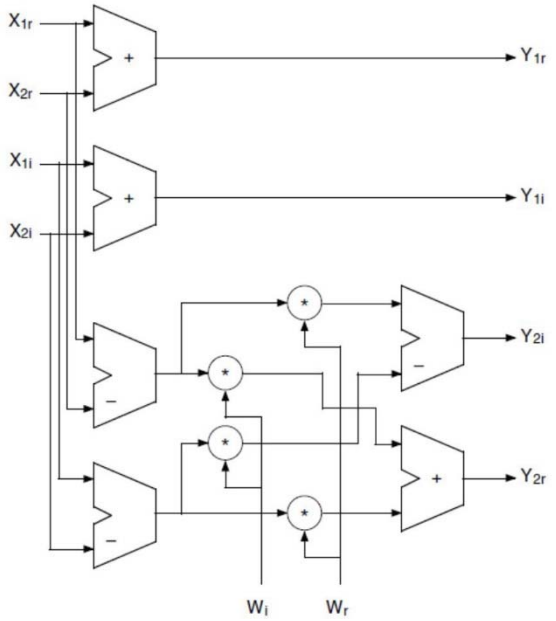



Fig. 6. All lines carry complex pairs of 32- bit IEEE-754 numbers and all operations are complex. The complex add, subtract, and multiply operations shown in Fig. 7 can be realized with a discrete implementation that uses two real adders to perform the complex add or subtract and four real multipliers and two real adders to perform the complex multiply. The complete butterfly consists of six real adders and four real multipliers as shown on the figure. In this and the following figure, all lines are 32-bits wide for the IEEE-754 single-precision data. Alternatively, as shown in Fig. 8, the complex add and subtract can be performed with two fused add-subtract units (marked as FAS in the figure) and the complex multiplication can be realized with two fused dot product units (marked as FDP). The area of the required standard cells and the worst-case delay



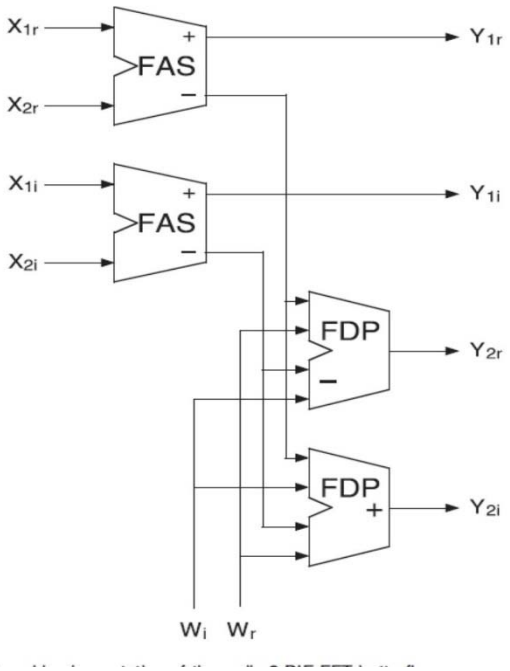

for the radix-2 butterfly units are shown in TABLE 4 Implementation of the Radix-2 DIF FFT Butterfly Unit

Table-4 implementation of the radix-2 DIF FFT butterfly unit

| Radix-2 Butterfly | Discrete | Fused |
|---|---|---|
| Standard Cell Area | 72,572 $\mu m^2$ | 47,489 $\mu m^2$ (65%) |
| Worst-Case Delay | 4.7 ns | 4.0 ns (85%) |

are obtained from having done a complete layout for the butterfly unit [11]. Thus, the areas are greater than the sum of the parts (i.e., four multipliers and six adders for the discrete version or two Fused DP and two Fused AS units for the fused version). The Fig. 5. Floating-point fused add-subtract unit (after [10]).

Table 2
Discrete (Parallel) and Fused Add-Subtract Area Comparison.
Table 3
Discrete Parallel and Fused Add-Subtract Comparison Fig. 6. Radix-2 DIF FFT butterfly. disparity is due to the clock distribution and I/O circuits. Similarly, the delays are less than the sum of the delays of the parts. This is because the conditions that produce the worst-case delay for one part are different from the conditions that produce the worst-case delay for other parts. In comparing the discrete and fused radix-2 butterfly units, the fused version requires about one-third less area and is about 15 percent faster than the discrete implementation. The area saving is the most significant, since in most signal processing applications,




pipelining can be employed to achieve higher throughput than that which might be expected from the inverse of the worst-case delay. For example, with a three stage pipeline either of the radix-2 butterfly designs can be clocked at over 500 MHz yielding a data rate of over 1 GSPS. The discrete implementation would be partitioned with the four input adders in the first stage, the multipliers in the second stage and the output adders in the third stage. The fused implementation would be partitioned with the Fused AS in the first stage and two stages for the Fused Dot Product (the first up through the carry save adder and the second stage for the addition, normalization, and rounding). Figs. 7 and 8 also provide information regarding the error characteristics. For the discrete implementation, rounding and normalization which may produce errors of as much as_1=2 of a least significant bit (¼2_23 for IEEE single precision floating point numbers), occurs at the output of each of the adders and multipliers. Thus, the components of butterfly output Y1 may have a 1=2 LSB error, while the components of output Y2 may have a 2 LSB error. For the fused implementation, rounding and normalization occurs at the output of the Fused AS unit and the Fused Dot Product. Thus, the components of butterfly output Y1 may have a 1=2 LSB error, while the components of output Y2 may have a 1 LSB error. These errors are the worst case limits.

**6 Radix-4 FFT Butterfly**

As a second example of the use of the new fused floating-point primitives, a radix-4 decimation in time FFT butterfly has been designed. The butterfly is shown in Fig. 9. All lines carry complex pairs of 32-bit IEEE-754 numbers and all operations are complex. The radix-4 butterfly consists of three complex multiplies and eight complex adds. A discrete realization of the parallel radix-4 FFT butterfly requires 12 real multipliers and 6 real adders to implement the 3 complex multipliers and 16 real adders to implement the 8 complex adders, for a total of 12 real multipliers and 22 real adders. The alternative fused realization requires 6 Fused Dot Product units to implement the 3 complex multipliers and 8 fused add-subtract units to implement the 8 complex adders, for a total of 6 Fused DP units and 8 Fused AS units

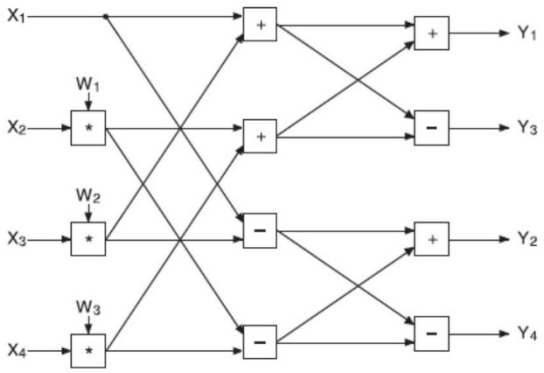



Fig. 7. Discrete implementation of the radix- 2 DIF FFT butterfly.

Fig. 8. Fused implementation of the radix-2 DIF FFT butterfly.

Fig. 9. Radix-4 DIT FFT butterfly.

Table 4

Implementation of the Radix-2 DIF FFT Butterfly Unit

Table 5 compares the placed and routed designs of the

TABLE 5
Implementation of the Radix-4 DIT FFT Butterfly Unit

| Radix-4 Butterfly | Discrete | Fused |
|---|---|---|
| Standard Cell Area | 250,099 $\mu m^2$ | 184,184 $\mu m^2$ (74%) |
| Worst-Case Delay | 6.9 ns | 6.0 ns (87%) |

TABLE 6
Errors in FFT Butterflies

| | Discrete | Fused |
|---|---|---|
| Radix-2 | ½ to 2 LSB | ½ to 1 LSB (60%) |
| Radix-4 | 2½ LSB | 1½ LSB (60%) |

discrete and fused implementations of the radix-4 decimation in time FFT butterfly. The fused radix-4 FFT butterfly design is about 26 percent smaller and 13 percent faster than the discrete parallel implementation. The block diagram of Fig. 9 allows estimation of the error characteristics. For the discrete implementation, rounding and normalization which may produce errors of as much as _1=2 of a least significant bit at the output of each of the multipliers and the following adders. All of the butterfly outputs may have errors of as much as 2 1=2 LSB. For the fused implementation, rounding and normalization only occurs at the output of the Fused Dot Product, and then once at each of the two layers of Fused Add-Subtract units. Thus, all of the fused butterfly outputs may have errors of as much as 1 1=2 LSB.

**7 Error comparison**

The previous sections have provided some sense of the error behavior of the discrete and fused implementations of the radix-2 and radix-4 FFTs. The worst case errors for the discrete and fused versions of the radix-2 and radix-4 butterflies are given in Table 6. For both radices, the fused implementations have

TABLE 7
Errors in 64K Point with Radix-2 FFT

| 64K point FFT | Discrete | Fused |
|---|---|---|
| Average Error | 0.047 | 0.038 (80%) |
| Maximum Error | 0.263 | 0.176 (76%) |

TABLE 8
Errors in 64K Point with Radix-4 FFT

| 64K point FFT | Discrete | Fused |
|---|---|---|
| Average Error | 0.039 | 0.026 (67%) |
| Maximum Error | 0.174 | 0.126 (72%) |

roughly 40 percent lower worst case errors. Since an N-point FFT uses Log rN stages, either k radix-4 stages or 2k radix-2 stages are required to compute a given length transform. Thus, even though the radix-2 butterfly error is less, the error in a transform is expected to be less for a radix-4 implementation. This was confirmed by performing a simulation of a 64K point FFT. First, a 64K point input vector consisting of random numbers was formed using RANDC. Then, an FFT was computed for the input using Mat lab double precision floating-point arithmetic to serve as the reference for subsequent results. The random number input vector was transformed with RTL models of a 64K point FFT using discrete and fused implementations of the radix-2 and radix-
4 FFT butterflies. The absolute differences between the reference and the discrete and fused results were evaluated for each of the 64K points. The average and maximum values of the errors are shown on Tables 7 and 8. For both the radix-2 and radix-4 based transforms, the fused implementations exhibit lower average errors and lower maximum errors. The fused implementations have about 25 percent less error than the discrete implementations. The radix-4 implementations have about 28 percent less error than the radix-2 implementations.

**8 conclusion**

This paper describes the design of two new fused floating-point arithmetic units and their application to the implementation of FFT butterfly operations. Although the fused add-subtract unit is specific to FFT applications, the fused dot product is applicable to a wide variety of signal processing applications. Both the fused dot product unit and the fused add-subtract unit are smaller than parallel implementations constructed with discrete floating-point adders and multipliers. The fused dot product is faster than the conventional implementation, since rounding and normalization is not required as a part of each multiplication. Due to longer interconnections, the fused add-subtract unit is slightly slower than the discrete implementation. The area of the fused radix- 2 butterfly is 35 percent smaller and the latency is 15 percent less than the discrete radix-2 FFT butterfly parallel implementation. The area of the fused radix- 4 butterfly is 26 percent smaller and the latency is 13 percent less than the discrete radix-4 FFT butterfly parallel implementation. Both fused butterflies use fewer rounding operations resulting in more accurate results than the discrete approaches. The errors for a 64K point FFT are about 25 percent less for the fused implementations.

**Acknowledgments**

Special thanks are due to the referees that have made several suggestions to significantly improve the paper.

**References**

IEEE Standard for Floating-Point Arithmetic, ANSI/IEEE Standard 754-2008, Aug. 2008. [2] R.K. Montoye, E. Hokenek, and S.L. Runyon, “Design of the IBM RISC System/6000 Floating-Point Execution Unit,” IBM J. Research and Development, 1990; 34:59-70.